International Journal of Engineering and Technical Research (IJETR)
ISSN: 2321-0869 (O) 2454-4698 (P) Volume-9, Issue-12, December 2019

A Brief Didactic Theoretical Review on Convolutional Neural Networks, Deep Belief Networks and Stacked Auto-Encoders

MSc.Rômulo Fernandes da Costa, MSc.Sarasuaty Megume Hayashi Yelisetty, Dr. Johnny Cardoso Marques, Dr. Paulo Marcelo Tasinaffo

***Abstract*—This paper presents a brief theoretical review on deep neural network architectures, deep learning procedures, as well assome of its possible applications. The paper focuses on the most common networks structures: Convolutional Neural Network (CNN), Deep Belief Network (DBN) and Stacked Auto-encoders (SA). The building blocks which enabled the construction of deeper networkssuch as Rectified Linear Unit (ReLU) and softmax activation functions, convolution filters, restricted Boltzmann machines and autoencoders, are explained in the beginning and middle sections of the paper. A few examples of hybrid systems are also presented at the last sections of the paper. The paper concludes with some considerations on the state-of-art work and on the possible future applications enabled by deep neural networks.**

***Index Terms*— Autoencoder, Boltzmann Machine, Convolutional Neural Network, Deep Leaning Review.**

I. INTRODUCTION

Neural networks algorithms have been applied to a wide variety of complex tasks, inareas ranging from computer vision, speech recognition, text translation, system identification and control, among others.

The greatest advantage of this algorithm lies on their ability to learn from a set of examples, without the need for defining a set of explicit rules for a given task. After learning how to solve a given problem, an artificial neural network would generally perform in the same level or better than a rule-based algorithm for the same task, especially for very abstract problems such as in computer vision.

While neural networks were shown to theoretically be able to represent any nonlinear function [1], in practice neural networks were limited in depth and by long training times. What allowedneural networks to achieve the high level of performance seen today was the development of a series of techniques for training deep networks in the past decade. This set of techniques is what is now known as deep learning.

This paper presents a brief theoretical review on deep neural network's structures, training procedures, and enumerates some of its possible applications. The paper focuses on presenting a general description on the inner workings of the most common deep architectures, namely the Deep Belief Networks (DBN), Stacked Autoencoders (SA) and Convolutional Neural Networks (CNN).

In terms of structure, these three topologies can be decomposed in fundamental blocks, such as the ReLU and softmax activation functions, convolution filters, restricted Boltzmann machines and autoencoders. These blocks, along with the associated architectures, are described in the middle sections of the paper.

A few examples of hybrid systemsare also presented at later sections of the paper. The paper concludes with some considerations on the state-of-art work and on the possible future applications enabled bydeep neural networks.

Rômulo Fernandes da Costa, GraduateProgram in ElectronicEngineeringAnd Computer Science, ITA, São José dos Campos

Sarasuaty Megume Hayashi Yelisetty, Graduate Program in Electronic Engineering And Computer Science, ITA, São José dos Campos, Brazil

Johnny Cardoso Marques, Computer Science Division, ITA, São José dos Campos, Brazil.

Paulo Marcelo Tasinaffo, Computer Science Division, ITA, São José dos Campos, Brazil.

This work was funded by the Brazilian National Council for Scientific and Technological Development (CnPq), in the form of funding for the first author.

II. BASIC CONCEPTS

A. *Artificial Neuron Structure*

An Artificial Neural Network (ANN) is a parallel computational structure loosely inspired on real neural networks,capable of learning from large amounts of data. The network is trained to generate a set of outputs from the inputs presents on the training data. Thus, an ANN can act as anuniversal approximator of nonlinear functions [1].

These networks are composed of several small units, called neurons or nodes, grouped in multiple sequential layers. Each neuron in a layer receivessignals fromneurons localized in other layers or from the network's input itself.The neuron then responds by emitting a signal of its own, propagating the information forward to the next layers in the network.

The output signal$y_n$fired by a neuron as a response to an input vector $\boldsymbol{x_n}$ isdescribed by:

$$y_n = \sigma_n(\boldsymbol{w_n x_n} + b_n) \quad (1)$$

Here, $\boldsymbol{w_n}$ and b_n are the connection weight vector and activation bias respectively.The mathematical function σ_n is a nonlinear function called "activation function" and describes the response of the neuron to its collective input.

Historically, σ_n used to be simple linearfunctions (such as in the original perceptron[2]) and sigmoid functions, but with the popularization of deeper networks, less computationally expensive options such as Rectified Linear Unit (ReLU) started to be employed. Fig. 1 shows a plot ofsome of the commonly used activation functions.

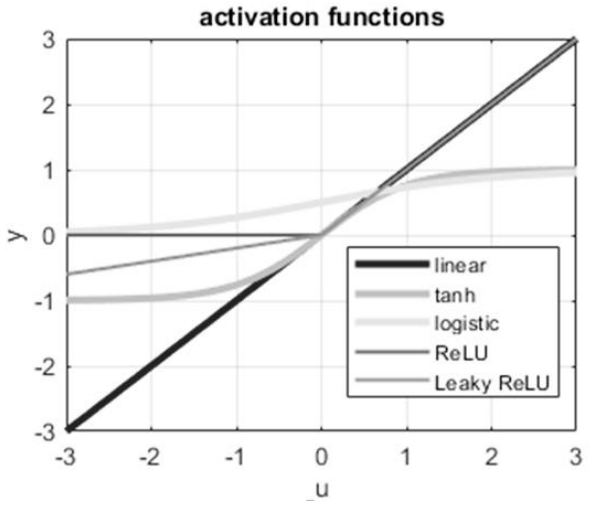

Fig. 1 – Common activation functions.

A special activation function is the "softmax" function, which normalizes the sum of outputs of all neurons of the previous layer to the interval of 0 to 1. The softmax output generatedby a neuron of raw output x_i is given by:

$$Softmax(\boldsymbol{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \qquad (2)$$

Where *n* is the number of possible classes considered by the Softmax layer.

This is useful for classification problems, where it is necessary to calculate the probability of the input to belong to a given *i* class, among all possible *n* classes [3].For example, assume that a softmax layer receives an input$x_i > 0$, while all other inputs $x_k$are zero. This will cause the layer to increase the probability of the *i* class and reduce the probability of all other *k* classes. The output of the network for this scenario is shown in Fig. 2.

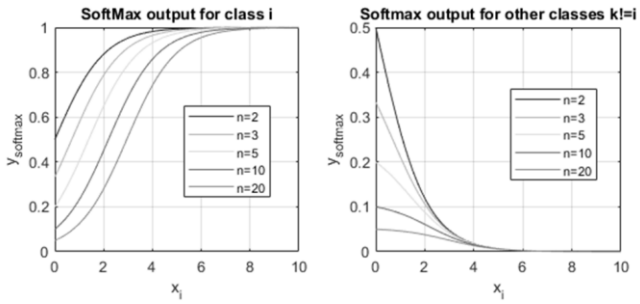

Fig. 2 – Softmax activation function.

B. *Feed-Forward and Recurrent Networks*

Neural Networks can be categorized in two groups depending on information flow direction: feedforward networks and recurrent networks.

Feedforward networksare designed so that input information is propagated layer to layer in a unidirectional manner.There are no feedback connectionsbetween themore advanced layersto theprevious layersin the network. This is appropriate for time invariant problems, as the network is not required to keep track of past events. Anexample is the multi-layer perceptron (MLP)[4], Convolutional [5] and Deep Belief networks [6]. An example of feedforward neural network is shown at the left side of Fig. 3.

A recurrent network (also called feedback network) allows information to also flow backwards in the network, allowing the network to remember paststates. This creates a nonlinear dynamical system [7] which can be trained to contextualize and retain some of the information already processed by the network, making the network more appropriate to tackle time variant problems.This added complexity comes with a cost. As the ANN's behaves as a highly nonlinear dynamical system, more complex learning algorithms are required to ensure the network'sstability [8]. Some examples of dynamical ANN are the Hopfield [4], Elman[9] and the Long Short-Term Memory (LSTM)[10] networks.An example of a recurrent neural network is shown at the right side of Fig. 3.

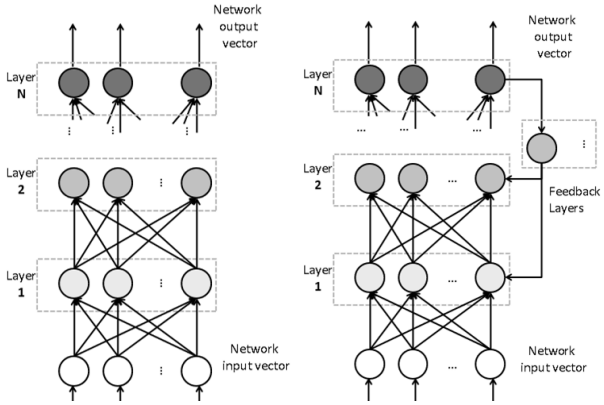

Fig. 3 – Example of feedforward and recurrent neural networks.

C. *Training as a nonlinear parameter estimation task*

The training procedure of an ANN, in which the networks learns to replicate a given function from data, can be thought as a parameter estimation problem.An ANN learns by changing the weights of the connections of its neurons, so as to minimize anestimation error (such as Mean Square Error, MSE) between the correct output and the output generated by the network. Due to the nonlinearity present in activation functions, directmethods such as least squares estimation cannotbe applied, and therefore, iterative methods are required[11].

The most common method for adjusting the weights in a neural network is the backpropagation algorithm and its variations, such as the Levenberg-Marquardt method [12]. The algorithm performs gradient descent over an error-measuring function, by applying the chain rule over each layer to find the appropriate adjustment for every weight present on the network.

For shallow networks with one or two hidden layers, it is possible to use backpropagation algorithms over the entire network, but for deeper architectures, the gradient either fades away over layers (referred as "vanishing gradient problem") or increases indefinitely (referred as "exploding gradient problem"), with the former problem especially prevalent on recurrent networks[13].

Another issue is data availability and reliability. For problems where data is often readily available but just as often mislabeled (for example, user generated content in video streaming websites), unsupervised learning methods(non-reliant on labeled data) are required.

In some other problems, training data is limited orfewer than the number of parameters to be adjusted.This enables the network to simply memorize the training data, generating an

exact copy of the output during training, rather than generalizing for the underlying model. This problem is referred as"overfitting" in the literature [13] [14].

To prevent these problems, several differentcomponents for building deeper nets were designed,such as restricted Boltzmann machines, autoencoders and convolution kernels, providing the foundation of what is now known as "deep learning". A common traitshared by all of these structures is that they provide means forextracting salient features contained in input data, generating a more abstract set of inputs to beprocessed by the next block. The training at this point is in most cases unsupervised. The final blocksprovide a highly abstract preprocessed data, which is then used for generating the network's output. The learning in this final stage is supervised, that is, uses the labeled data in the training set. The following sections describe these structures in more detail.

III. Deep Belief Neural Networks

A. Boltzmann Machines

A Boltzmann Machine (BM) is a network model introduced by Hinton,Sejnowski and Ackley in 1985[15]. This structure resemblesa Hopfield network, in the sense that both are capable of learning the internal distribution of a training set. However, aRestricted Boltzmann Machine (RBM)uses a stochastic activation function based on the Boltzmann distribution, and its training is inspired on thermodynamic principles.

This network comprises a visible input layer and a hidden layer, with each neuron being connected to all other neurons in the network. The visible layer contains neurons whose states can be directly overridden by an input signal, while hidden layer comprises neurons which can only be accessed by the visible neurons. This topology is shown at the left side of Fig. 4.

Each *i*-th neuron has two states, "on"($s_i = 1$) or "off"($s_i = 0$). The networks internal energy *E*is given by:

$$E = -\sum_{i<j} w_{ij} s_i s_j + \sum_i \theta_i s_i \quad (3)$$

Where θ_i is bias of the*i*-th neuron, and w_{ij} is the connection weight between the *i*-th and *j*-th neurons. The energy differenceΔE_ibetween "off" and "on" states of the *i*-th neuron is given by:

$$\Delta E_i = \sum_j w_{ij} s_i - \theta_i \quad (4)$$

The probabilityp_i of the *i*-th neuron setting its state to "on" follows the Boltzmann distribution,hence the network's name.

$$p_i = \frac{1}{(1+e^{-\Delta E_i/T})} \quad (5)$$

Where *T* is analogous to system temperature. It can be noticed that (3) (4) and (5) model a system steadily losing energy, eventually reaching a thermal equilibrium state.

BM training has the goal of minimizing the difference between the probability distribution of the state ofvisibleneuronsat thermal equilibrium, P^-, and the probability distribution of the training samples, P^+.The difference between two distributions can be measured by the Kullback-Leibler's (KL) divergence, *G:*

$$G = \sum_v P^+(V) \ln\left(\frac{P^+(V)}{P^-(V)}\right) \quad (6)$$

Where *V* denotes each one of the visible input neurons. Fortunately, the gradient of G in relation to any given weight w_{ij} has a very simple form:

$$\frac{\partial G}{\partial w_{ij}} = -\frac{1}{T}(p_{ij}{}^+ - p_{ij}{}^-) \quad (7)$$

Where $p_{ij}{}^+$is the probability of both *i*-th and *j*-th being "on" at thermal equilibrium, with all input neuronsoverridden by a training signal, while$p_{ij}{}^-$ denotes the probability of the same event happening without the influence of the training signal.

This indicates that a simple rule can be applied to change the weights. Assuming a step size of ε, the variation on weights $\Delta w_{ij}$and on biases $\Delta\theta_i$is given by:

$$\Delta w_{ij} = \varepsilon(p_{ij}{}^+ - p_{ij}{}^-) \quad (8)$$
$$\Delta\theta_i = \varepsilon(p_i{}^+ - p_i{}^-) \quad (9)$$

While (8) and (9) are fairly simple learning rules, the excessive complexity of the BM's topology causes the learning to be inefficient for complex problems. Likewise, the training requires that the BM is simulated until it reaches thermal equilibrium, which increases training complexity [16].

An improvement on training performance can be achieved restricting BM's topology, creating a so-called restricted Boltzmann Machine (RBM), with the most common restriction being eliminating connections between neurons in the same layer [17]. This RBM is shownat the right side of the Fig. 4.

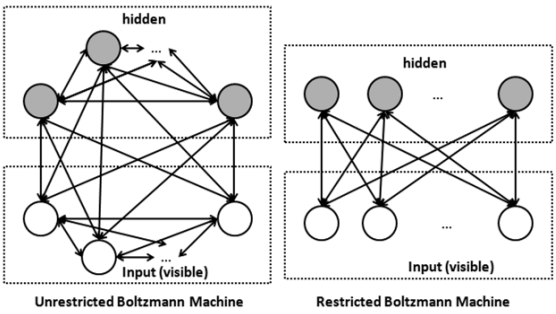

Fig. 4 – Unrestricted and Restricted Boltzmann Machines

The neuron state determination process is also simplified as a result of the topology restriction. The probability of a givenneuron (visible or hidden) being active, given the current state of the neurons on the opposite layer, can now be written similarly to (1).

$$p(v_i = 1|\boldsymbol{h}) = \sigma(a_i + \sum_j h_j w_{ij}) \quad (10)$$
$$p(h_j = 1|\boldsymbol{v}) = \sigma(b_j + \sum_i v_i w_{ij}) \quad (11)$$

Where $p(v_i = 1)$ and $p(h_j = 1)$ are the probability of a visible or a hidden layer being on the active state, w_{ij} is the connection weight between neurons *i*and *j*, and a_i and b_j are biases for the visible and hidden neurons respectively. In the RBM's case, σ is a logistic function.

The learning function can also be simplified, as to not depend on simulating the machine to its thermal equilibrium. Instead, the RBM can be simulated for a *k* number of cycles, with the state of all neurons (including those on the hidden layer) being sampled for the learning rule.

Initially, the visible neurons in the RBM are overridden by with an input vector contained in the training set, so thatall hidden neurons change theirstate accordingly to (11). The state of all neurons is represented by $< v_i h_j >_{data}$. The

visible neurons are then unclamped, sothat the network isfree toreconstruct the input vector using the information stored in the hidden layers. The neurons are then updated sequentially, with their states being set accordingly to (10) and (11). This procedure is called Gibb's sampling. After *k* cycles simulating the network by Gibb's sampling, the state of the neurons $< v_i h_j >_{rebuilt}$ is then sampled. The weight adjustment is given by:

$$\Delta w_{ij} = \varepsilon(< v_i h_j >_{data} - < v_i h_j >_{rebuilt}) \quad (12)$$

This algorithm is called Contrastive Divergence, or CD-*k* [18]. It is much faster learning algorithm, as it does not require simulating the RBM to its equilibrium point. Even though the learning rule in (12) does not follow the divergent *G*defined on (6), it stillprovides a sufficiently precise approximation for training the RBM, even for a single simulation cycle (*k=1*) [19].

B. *Deep Belief Networks (DBN)*

Deep BeliefNetworks (or more commonly Bayesian networks) are statistical models that represent a set of variables along with their conditional dependencies. This network can infer the probability of an event being caused by one of several possible known causes.An example of this structure is shown in Fig. 5, with each event's dependency on others is marked with arrows.

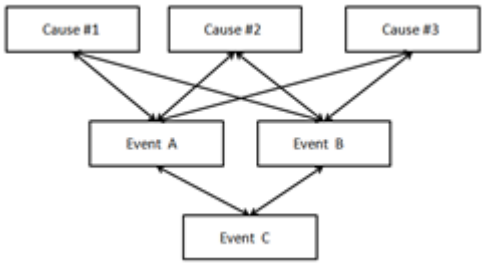

Fig. 5 – Belief network general structure.

This network can infer variables, parameters and hidden variables from the distribution of a given event. Thiscapacity for inference and general structure has been perceived as similar those found onBoltzmann Machines in the 1980's[14], but due to the inherentcomplexity of BM's topology and its training, this similarity could not be further explored until two decades later.

The creation of RBM's in the late 1980's [16] andthe development of faster algorithms for their training in early 2000's [18] led to development of deep belief nets (DBN)composed of stacked RBMs. The proposed structure is seen on Fig. 6.

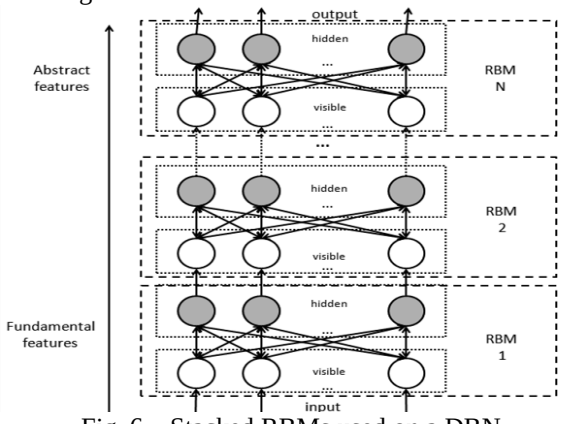

Fig. 6 – Stacked RBMs used on a DBN.

The structure presented in Fig. 5 shows a stack of*N*RBMs. The visible layer of the bottom RBM is used as the input layer of the DBN, with the hidden layer acting as input for the next RBM. All RBMs are connected in the same way, forming a sequential structure up to hidden layer of the last RBM, which providesthe output of the stack.

Each one the RBMs are trained separately in an unsupervised manner, from the bottom RBM's to the top. After finishing training an RBM, their hidden neurons will register some of the most salient characteristics contained on its training inputs. The output of the trained RBM is then used as a training input for the next RBM in the stack, and so on. As a result, each RBM extracts a feature present in the training data, with higher layers extracting increasingly more abstract information [20].

After training all RBMs in the stack, the output from the top layer can be used to activate a classifier network using the *softmax*function. The training in this stage is now supervised, as the labeled datais associated with the features extracted by the stack[21].

IV. Stacked Autoencoders

A. *Autoencoders*

Another building block which can be stacked to form a deep network is the autoencoder, also known in literature as anauto associator. An auto-encoder is a neural network designed for creating an efficient representation of its input data, often with reduced dimensionality. Like the RBM, it can be used for extracting features from an unlabeled set of data, although their training procedures are quite different [22].

The traditional autoencoder is a simple network trained to regenerate its own input as its output, hence the structure's name. Learning can be done through the standard backpropagationalgorithm, in the same way as the MLP.As the training goal is to learn its own input value,it does notrequire labeled data, making it a form of unsupervised learning.

After the training is completed, the hidden layers of the autoencoder will have learned a useful encoding of the data, based on the distribution of the training data. Likewise, the output layer will also learn a form to decode and regenerate the original data from its encoding [23].

A common issue of the traditional autoencoder's training is that the network has the possibility of learning the identity function, preventing the autoencoder from learning a more useful representation of the data.

To avoid this problem, a few variations of the autoencoder have been designed, such as denoising autoencoders andsparse autocoders. Denoising autoencoders explicitly introduces noise in the input's training set, so that the network is trained to filter the noise present in the input[24].The sparse autoencoder introduces an additional penalty during training to constrainhidden neurons activations and weights to a set of predetermined values, preventing the network from learning the identity[25]. Both techniques forces the autoencoder to rely on features contained in the training data.

A simpler solution for the issue is by having a smaller number of hidden neurons than inputs in the autoencoder, creating an under-complete system.This prevents the network from learning the identity function, although at the cost of learning less features compared with over-complete autoencoders [21].

An illustrated example of a MLP network trained as an autoencoder is shown on Fig. 7. After training the MLP either traditionally or using any of the variants presented, the autoencoder's input x is encoded by the hidden layer, creating aencoding function $x' = f_{encoding}(x)$. The output layerapproximatelyreconstructs the original input as $y \cong x$.

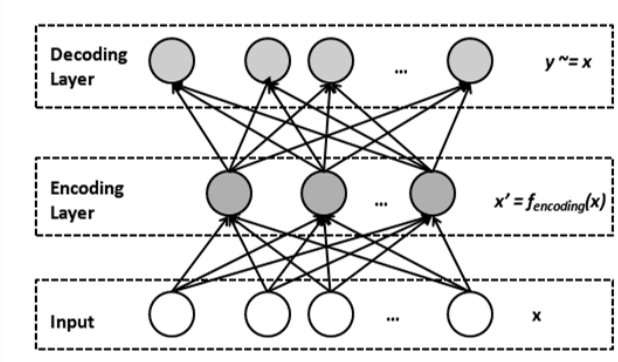

Fig. 7 – Example of an autoencoder

B. *Stacked Autoencoders*

Autoencoders can be stacked to create a single deep network,similarly to how RBMs are used on a DBN. The first autoencoder is trained to encode and regenerate the network's input. After training, the decoding layer is discarded, and the hidden layers' output (the encoded information x') is used as input to train the next autoencoder. This procedure is repeated for each autoencoder in succession. After all autoencoders in the stack have been trained, the stack is then connected to the input of a fully connected network, which will generate the output of the network. Finally, all layers are then fine-tuned using supervised training[13].

Fig. 8 shows the structure of a deep network formed by the stacked autoencoders (SA). While this is very similar to DBN in structure, SA composed of traditional auto-encoders has slightly inferior results compared to DBN[26] due to the CD-k algorithm being a better approximation of log-likelihood gradient than the reconstruction error gradient, while the denoising variation of the autoencoder performed similarly to DBN [13].

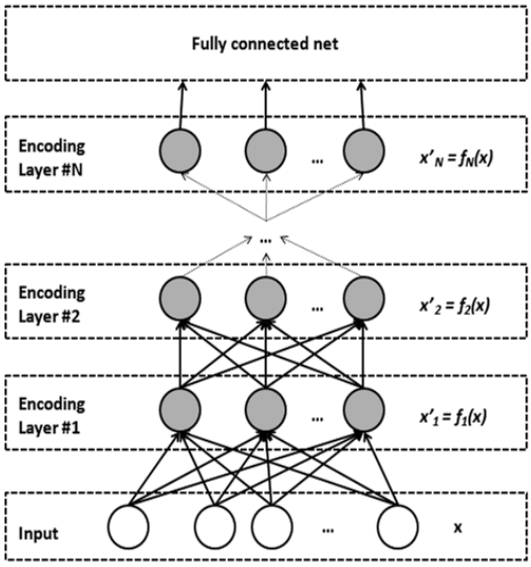

Fig. 8 – Stacked autoencoder structure

V. Convolutional Neural Networks (CNN)

A. *CNN development history*

Both methods for creating the deep neural nets presented previously, DBN and SA, make use of greedy layer-wise training to avoid the vanishing gradient problem. Until 2006, there were no means for training deep networks withfully connectedlayers.

However,a specialized type of deep network that could be trained traditionally was developed two decades prior to the appearance of DBNs. These networks are the Convolutional Neural Networks (CNN or alternatively ConvNet). They have a fanned-out structure, which simplify its training.

Convolutional neural networks follow an architecture loosely inspired on the visual system of living beings. The works of Hubel and Wiesel, at the end of the 1950s and through the 1960s, gave start to this line of research by presenting the response generated by the striate cortex of cats to several moving stimuli of light [27].

In their experiment, it was found that some isolated neurons (referred by the authors as "simple" cells) fired only when subject to image edges in veryspecific positions and orientations, while some others neurons with larger receptive fields (referred as "complex" neurons) reacted specifically to more complex stimuli, whilebeing insensitive to changes in position. These properties had been verified in the nervous system of cats[26] and monkeys [28].

The concept that an image could be processed by several parallel processing units, each one absorbing increasingly abstract features of the image inspired the development of the "neocognitron" network by K. Fukushima, which is considered to be the first example of CNN in literature [29]. The neocognitron is composed of several sequential modules, with each module comprising two layer of neurons. The first layer contains unitsthat resemblethe simpler neurons described by Hubel and Wiesel, which are responsible for extracting edges from the image.The second layer contains units that resemble the complex neurons, which are less sensitive to position. The complex neurons' outputs feed the next module, creating a serialized structure similar to DBNs and SA. Another similarity with modern networks is that the Neocognitron uses unsupervised training to learn features present in the data.

The work inspired the design of many other networks based on this hierarchicalarchitectures, such as the convolutional network developed by LeCun in 1989 for recognizing hand-written digits [30].

This networkaccepts as input an image of 16x16 pixel andcontains three hidden layers.A single output layer classifiesthe ten possible digits in the input image.

The first hidden layer has 12 convolution kernels, each one composed of 64 neurons, which scans the image for local features. By performing spatial convolution of the input with each kernel,twelve feature maps of 8x8 pixelsare obtained.

The second layer has 12 convolution kernels, each one with 16 neurons, creating a 4x4 pixel feature map. The third layer had 30 neurons and processed the more abstract features contained in the image. Backpropagation was applied to learn all connection and kernel weights.

Further work on this network led to the development of Lenet-5 by the same author in 1998[31], which had a profound impact in the field of computervision. Lenet-5 has 7 layers and accepts a 32x32 pixel image as its input.

The first layer extracts 6 feature maps of 28x28 pixels,

obtained by performing convolution of the input with a 5x5 pixel kernel. The second layer subsamples the feature maps into smaller feature maps of 14x14 pixels, by subdividing the feature maps into groups of 2x2 pixels and averaging their values.

The third layer extracts 16 feature maps of 10x10 pixels, byperforming convolutions with5x5 pixel kernels. The 16 feature maps arethen subsampled by the fourth layer in the same way as the second layer, generating 16 feature maps of 5x5 pixels.

The fifth layer extracts 120 feature maps of 1x1 pixels, by convoluting its input with 5x5-pixel kernels. Due to the size of the resulting feature map (composed of a single pixel), this layer acts as a fully connected layer.

The sixth layer has 84 neurons. This number has been chosen so that its output can be rearranged to form a 7x12-pixel image. The seventh and final layer classifies the network's input, through Euclidean Radial Basis Functions (RBF) units. There is one RBF unit for each possible class, with the centers hyper-parameters of each RBF purposely defined to compare the image generated by the sixth layer with all of the possible classes.

LeNet 5 was superior to many other character identification algorithms of its time. This topology based onsequential convolutions and sub sampling layers is still used in modern CNNs.In fact, the best algorithms for object identification on computer vision are based on CNNs, with examples such as GoogleLeNet, a 22-layer deep CNNwhich achieved a classification error of about 6% on ImageNet's Large-Scale Visual Recognition Challenge in 2014[32][33]. The use of GPUs for training CNNs, a technique introduced in the 2000's,greatly reduced learningtimes [34][35], further popularizing this architecture.

B. *CNN structure*

Current CNNs are composed of several layers reducing in size, with each layer applying a form of compression on its input data. The neurons on these layers are not fully connected to the other layers, greatly reducing the number of weights to be adjusted by training. Data is reduced in size from layer to layer, until the final layers.The final layers are usually densely connected, so as to usethe compressed data to perform classification or regression. An example of typical CNN structure used for classifying images is shown in Fig. 9.

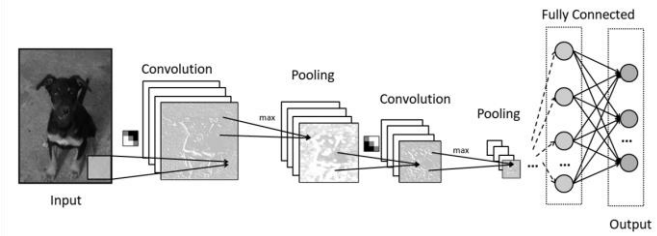

Fig. 9 – Typical CNN structure for image classification

The initial compression stage is composed of two types of layers, convolution layers and pooling layers. Convolution layers are the main building block in CNNs. The layer's input is swept by a filter (the convolution kernel) designed to extract a specific feature. This extraction is done by performingcross-correlation (convolution over a mirrored kernel) of the input with the kernel, centered on the first element of the input. After each convolution, the kerneladvancesa number of elements on the input before executing the next convolution.The number of elements advanced is called "stride". This procedure is repeated until the entire input is swept by the kernel. A ReLU function may be used after each convolution to increase nonlinearity in activation [36].

The second type of layer on the compression stage of CNNs the pooling layer. These are usually placed after a number of convolution layers. Pooling refers to down sampling the input by partitioning it into smaller groups of elements, and then performing anonlinear function over the subgroup elements, called the pooling function. The output of the pooling function for each subgroup is used as the layer's output. Historically, averaging and *tanh* was used as pooling function as seen in LeNet-5[30], but recentCNNs use*max*, as it can converge faster and have better generalization in practice [37].

Fig. 10 shows the results of an example image convoluted by four 3x3 pixel kernels, followed by a 2x2 pixel max pooling. The white squares represent an element of value*1*, gray squares represent*0* and black squares *-1*. Notice that each kernel extracts a different set of edges from the image.

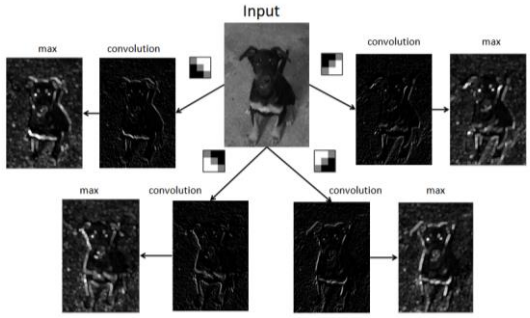

Fig. 10 – Image processed by convolutional kernels and max operations.

The second stage performs generates the output of the network using the data acquired by the first stage. The final layers are fully connected, with large numbers of adjustable parameters. To improve learning speed and preventing neuron saturation, ReLU is often used as activation function.Another problem caused by the large number of weights is overfitting, as the number of examples is often outnumbered by the number of parameters in several applications.

A recent developedtechnique for preventing overfitting is "dropout". This technique consists in randomly deactivating neurons during training with probability *p*. This forces the network to not rely on a few specific neurons, therefore improving network robustness. After training, all neurons may be used normally, buttheir outputs(or their corresponding weights) should be multiplied by a factor *1-p* to compensate for the deactivation procedure[38].

VI. Hybrid Networks

A. *Generative Adversarial Networks (GAN)*

The three networks presented previously can be combined with other topologies and artificial intelligence techniques in order to create a larger hybrid network. This is done to allow domain-specific networks to function beyond their original design purposes and limitations. The most common hybrids combine CNN architectures with other technologies, justified by the CNN's simplicity and impact in the computer vision community.

A recent example is Generative Adversarial Networks (GAN) created in 2014 [39]. This network combines two smaller networks acting in opposing roles, as a generative network and a discriminator, often in a form of zero-sum game. Differently from the majority of other deep networks, which are focused on classification problems, GANs are primarily used as a generative model, and are often used in practice for generating synthetic images [40].

The GAN's generative network, *G*, is trained to generate data samples that match thedistribution presenton the training set, similarly to how a Boltzmann machine is trained to generate new data. The second network, the discriminator*D*, is trained to identify the synthetic data created by the generative network among the real examples contained in the training set. This discriminator is often a CNN.

Both networks are trained alternately, in a form so that the loss functionfor *G*and *D*canbe rewritten to represent aminmax game. Training is often done using gradient descent algorithms, but due to the nature of the game between *G* and *D*, the algorithm may fail to converge, as improvement of one network may incur a decrease on the adversary performance [41].

This procedure drives both networks into becoming better at generating and detecting the synthetic data. The training proceeds until the synthetic data created by *G* is undistinguishable from the training data.

B. RBM –CNN hybrids

Due to its generative capability and feature extraction properties, RBMs are often paired with CNNs and other Artificial Intelligence (AI) techniques.

A structure proposed in [42] is a CNN-RBM hybrid, for face verification in wild conditions. The last stages of CNN can be connectedto stack of RBMs, creating a form ofDBN on top of the convolutional stages. The method is shown to have comparable results with other high precision methods, with the advantage of not requiring hand-made filters for extracting features and strong alignment.

Another example of a CNN-RBM hybrid is shown in [43] for speech recognition. The structure presented in this work is a CNN that uses RBMs as the convolutional filters, creating a stochastic convolution layer. The RBMs can be pretrained to learn the correct filters. This improves network performance if the training set is small.

C. Autoencoder hybrids

Autoencoders hybrids are not as common as RBM and CNN hybrids, as the classical forms of the autoencoder lack some of the interesting properties of the former topologies. However, a recentvariation of the autoencoder called variational autoencoder (VAE) can act as generative model, which is a desirable property for creating hybrid systems.

A VAE combines variational inference and deep learning in a direct probabilistic graphical model. Rather than directly mappingthe input valueslike in the classical autoencoder's training, the VAE's trainingaims to model the parameters of the training set distribution, treating the set of inputs as probabilistic values.The training tries tominimize the KL divergence between the training input set and the reconstructed set.

After training, a VAE will learn not only the mean value of a given input, but also its variance. Thus, it is able to regenerateinputs though stochastic procedures[44]. This property can be used in conjunction with a CNN and recurrent models to generate text as presented in [45]. Moreover, a VAE can act as generative network for use in a GANs, as reported in [46].

VII. CONCLUSION

This paper has described the basic building blocks which constitute what is now known as Deep Learning, as well as some of the most common network architectures that can be built with those blocks.

The increase of depth allowed neural networks to achieve much higher levels of precision as a function approximator compared to their shallow counterparts. While any shallownetworks capable of approximating a function given enough neurons, the number of neurons required for a precise approximation increases exponentially as a result of the curse of dimensionality. The addition of extra layers in the network leads to a higher level of abstraction in data processing, and therefore higher precision, as seen on all of the architectures shown in the paper.

As a result of this increase on precision, deep neural networks have been used as solution for many problems for which there's no closed analytical solution, with many networks achieving much better performance over traditional rule-based algorithms.With the increase on data availability and hardware capacity, it should be expected that many of the rule-based algorithms used today for tacklingcomplex problems to be improved or even be superseded by neural-network based approaches in the near future.

ACKNOWLEDGMENT

We'd like to thank Dr. Osamu Saotome for his advice during the preparation of this article.

REFERENCES

[1] K.Hornik, M. Stinchcombe, and H. White. "Multilayer feedforward networks are universal approximators".in Neural networks,2(5), 1989, pp-pp 359-366.
[2] F. Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." Psychological review 65.6,1958, 386.
[3] I. Goodfellow, Y.Bengio, andA.Courville."Deep learning". MIT press, 2016.
[4] R.P. Lippmann. "An introduction to computing with neural nets." IEEE Assp magazine Vol. 4, no.2 1987. pp 4-22.
[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86, no. 11,1998. pp. 2278-2324.
[6] G.E Hinton, S. Osindero, and Y.W. Teh. "A fast learning algorithm for deep belief nets." Neural computation 18, no. 7, 2006, pp. 1527-1554.
[7] J. Singh andN.Barabanov. "Stability of discrete time recurrent neural networks and nonlinear optimization problems". Neural Networks, 74, 2016, pp.58-72.
[8] J.N. Knight, "Stability analysis of recurrent neural networks with applications". Fort Collins, CO: Colorado State University.2008.
[9] Y. Cheng, Wei-Min Qi, and Wei-you Cai. "Dynamic properties of Elman and modified Elman neural network." Proceedings. International Conference on Machine Learning and Cybernetics. Vol. 2. IEEE, 2002.
[10] S.Hochreiter and J.Schmidhuber. "Long short-term memory." Neural computation, vol 9 no.8 1997, pp. 1735-1780.
[11] B. J.T Morgan. "Non-Linear Parameter Estimation- an Integrated System in Basic." Journal of the Royal Statistical Society: Series C (Applied Statistics) 37, no. 3,1988, pp. 449-450.
[12] M.T. Hagan, and M. B. Menhaj. "Training feedforward networks with the Marquardt algorithm." IEEE transactions on Neural Networks 5, no. 6, 1994. pp. 989-993.
[13] Y. Bengio. "Learning deep architectures for AI." Foundations and trends® in Machine Learning 2, no. 1. 2009, pp. 1-127.

[14] S. Venkataraman, D. Metaxas, D. Fradkin, C.Kulikowski, and I. Muchnik. “Distinguishing mislabeled data from correctly labeled data in classifier design”. In 16th IEEE International Conference on Tools with Artificial Intelligence, November 2004, pp. 668-672.
[15] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. "A learning algorithm for Boltzmann machines." Cognitive science 9.1 1985, pp. 147-169.
[16] A. Fischer and C. Igel. "Training restricted Boltzmann machines: An introduction." Pattern Recognition 47.1,2014, pp. 25-39.
[17] P. Smolensky. “Information processing in dynamical systems: Foundations of harmony theory.” No. CU-CS-321-86. Colorado Univ at Boulder Dept of Computer Science, 1986.
[18] G.E. Hinton. "A practical guide to training restricted Boltzmann machines." In Neural networks: Tricks of the trade, pp. 599-619. Springer, Berlin, Heidelberg, 2012.
[19] G.E. Hinton. "Training products of experts by minimizing contrastive divergence." Neural computation 14, no. 8,2002, pp. 1771-1800.
[20] G.E. Hinton, S. Osindero, and Y.W. Teh. "A fast learning algorithm for deep belief nets." Neural computation 18, no. 7. 2006,pp. 1527-1554.
[21] A. Mohamed, G. Dahl, and G. Hinton. "Deep belief networks for phone recognition." In: Nips workshop on deep learning for speech recognition and related applications, vol. 1, no. 9, 2009. p. 39.
[22] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, andP.A. Manzagol. “Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion”. Journal of machine learning research, 11(Dec) 2010, pp. 3371-3408.
[23] S.P. Luttrell. “Hierarchical self-organizing networks”. In: Proc. 1st IEE Conf. Artificial Neural Networks. British Neural Network Soc., 1989. pp. 2-6.
[24] D. Erhan, P.A. Manzagol, Y. Bengio, S.Bengio, andP. Vincent. “The difficulty of training deep architectures and the effect of unsupervised pre-training.” In Artificial Intelligence and Statistics,2009, pp. 153-160.
[25] A. Ng. “Sparse autoencoder”. CS294A Lecture notes, 72, 2011,pp 1-19.
[26] Y. Bengio, P. Lamblin, D. PopoviciandH. Larochelle. “Greedy layer-wise training of deep networks”. In Advances in neural information processing systems, 2007, pp. 153-160.
[27] D.H. Hubel, andT. N. Wiesel. “Receptive fields, binocular interaction and functional architecture in the cat's visual cortex.” The Journal of physiology, 160(1), 1962, pp 106-154.
[28] D.H. Hubel, andT.N. Wiesel. “Receptive fields and functional architecture of monkey striate cortex”. The Journal of physiology, 195(1), 1968,pp. 215-243.
[29] Fukushima, K. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. Biological cybernetics, 36(4), 1980,pp. 193-202.
[30] Y. LeCun, B. Boser, J.S Denker, D. Henderson, R.E. Howard, W. Hubbard, andL.D. Jackel,. “Backpropagation applied to handwritten zip code recognition”. Neural computation, 1(4), 1989, pp 541-551.
[31] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. “Gradient-based learning applied to document recognition”. Proceedings of the IEEE, 86(11), 1998,pp. 2278-2324.
[32] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, and A. Rabinovich. “Going deeper with convolutions”. In: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 1-9.
[33] O. Russakovsky, et. Al. "Imagenet large scale visual recognition challenge." International journal of computer vision 115.3. 2015, pp. 211-252.
[34] K. S. Oh, and K. Jung. "GPU implementation of neural networks." Pattern Recognition 37.6, 2004, pp 1311-1314.
[35] K. Chellapilla, S. Puri, and P. Simard. "High performance convolutional neural networks for document processing." 2006.
[36] V. Romanuke. “Appropriate number and allocation of ReLUs in convolutional neural networks”. NaukoviVisti NTUU KPI, (1), 2017, pp. 69-78.
[37] D. Scherer, A. Müller, andS. Behnke.“Evaluation of pooling operations in convolutional architectures for object recognition.” In International conference on artificial neural networks. Springer, Berlin, Heidelberg. 2010, September, pp. 92-101.
[38] A. Krizhevsky, I. Sutskever, andG.E. Hinton. “Imagenet classification with deep convolutional neural networks”. In Advances in neural information processing systems. 2012, pp. 1097-1105.
[39] I. Goodfellow et al. “Generative adversarial nets.” In: Advances in neural information processing systems. 2014. pp. 2672-2680.
[40] T.C. Wang, M.Y. Liu, J.Y. Zhu, A.Tao, J. Kautz, and B. Catanzaro. “High-resolution image synthesis and semantic manipulation with conditional gans”. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2018,pp. 8798-8807.
[41] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved techniques for training gans”. In Advances in neural information processing systems, 2016. pp. 2234-2242.
[42] Y. Sun, X. Wang, andX. Tang.“Hybrid deep learning for face verification.” In Proceedings of the IEEE international conference on computer vision, 2013, pp. 1489-1496.
[43] O. Abdel-Hamid, L. Deng, and D. Yu. “Exploring convolutional neural network structures and optimization techniques for speech recognition.” In Interspeech,Vol. 11,2013, pp. 73-5.
[44] J.An, and S.Cho. “Variational autoencoder based anomaly detection using reconstruction probability”. Special Lecture on IE, 2(1), 2015.
[45] S.Semeniuta, A. Severyn, and E. Barth. “A hybrid convolutional variational autoencoder for text generation”. arXiv preprint arXiv:1702.02390, 2017.
[46] L. Mescheder, S. Nowozin, and A. Geiger. “Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks”. In Proceedings of the 34th International Conference on Machine Learning-Volume 70. 2017, August pp. 2391-2400. JMLR. org.

Rômulo Fernandes da Costa received his B.S. in Electrical Engineering from Sao Paulo State University (UNESP),and received his MsC in Electronic Engineering from Aeronautics Institute of Technology (ITA). He is currently a PhD candidate at ITA, contributing with the Laboratories of Space Robotics and Radar Signal Processing. His main areas of research are space robotics, machine learning and radar signal processing.

Sarasuaty Megume Hayashi Yelisetty received her B. S. in Computer Engineering from Vale do Paraiba University (UNIVAP), and received her MsC in Computer Engineering from Aeronautics Institute of Technology (ITA). She is currently a PhD candidate at ITA. Additionally, she has been working at EMBRAER in software/airborne electronic hardware processes during the last 8 years and has recognized experience in standards used for airborne system and software such as DO-178B/C, DO-254 and MIL-STD-498.

Johnny Cardoso Marques received the B.S. in Computer Engineering from University of the State of Rio de Janeiro (UERJ), the M.Sc. (in Aeronautical Engineering) and a PhD. (in Electronic and Computer Engineering) both from Aeronautics Institute of Technology (ITA). He is a current professor in the Aeronautics Institute of Technology (ITA). Additionally, he worked at EMBRAER in software processes definition for 15 years and has recognized experience in standards used for airborne systems and software such as DO-178C, DO-254, ARP-4754 and DO-200B. He is also part of several committees in IEEE Standards Association.

Paulo Marcelo Tasinaffo is graduated in Mechanical Engineering from the Federal University of Itajubá/MG (UNIFEI, 1996), obtaining a Master's degree in Mechanical Engineering from the same Institution in 1998. In 2003, he received a Doctorate in Space Engineering and Technology from the National Institute of Space Research (INPE) in Brazil. He is currently a full professor of the Aeronautical Technological Institute (ITA) in São José dos Campos/SP also in Brazil, with experience in Aerospace and Computer Engineering and with emphasis on Artificial Neural Networks. The Prof. Tasinaffo works mainly in the following subjects: modeling of non-linear dynamic systems, computational mathematics, artificial intelligence, expert systems, intelligent agents, neural control structures, evolutionary computation and stochastic processes.